N

Naïve Bayes

Geoffrey I. Webb
Monash University, Melbourne, Victoria

Synonyms

Idiot's bayes; Simple bayes

Definition

Naïve Bayes is a simple learning algorithm that utilizes ►Bayes rule together with a strong assumption that the attributes are conditionally independent, given the class. While this independence assumption is often violated in practice, naïve Bayes nonetheless often delivers competitive classification accuracy. Coupled with its computational efficiency and many other desirable features, this leads to naïve Bayes being widely applied in practice.

Motivation and Background

Naïve Bayes provides a mechanism for using the information in sample data to estimate the posterior probability $P(y \mid \mathbf{x})$ of each class y, given an object $\mathbf{x}$. Once we have such estimates, we can use them for ►classification or other decision support applications.

Naïve Bayes' many desirable properties include:

- *Computational efficiency*: ►Training time is linear with respect to both the number of ►training examples and the number of ►attributes, and ►classification time is linear with respect to the number of attributes and unaffected by the number of training examples.
- *Low variance*: Because naïve Bayes does not utilize search, it has low ►variance, albeit at the cost of high ►bias.
- *Incremental learning*: Naïve Bayes operates from estimates of low order probabilities that are derived from the training data. These can readily be updated as new training data are acquired.
- *Direct prediction of posterior probabilities*.
- *Robustness in the face of noise*: Naïve Bayes always uses all attributes for all predictions and hence is relatively insensitive to ►noise in the examples to be classified. Because it uses probabilities, it is also relatively insensitive to noise in the ►training data.
- *Robustness in the face of missing values*: Because naïve Bayes always uses all attributes for all predictions, if one attribute value is missing, information from other attributes is still used, resulting in graceful degradation in performance. It is also relatively insensitive to ►missing attribute values in the ►training data due to its probabilistic framework.

Structure of Learning System

Naïve Bayes is based on ►Bayes rule

$$P(y \mid \mathbf{x}) = P(y)P(\mathbf{x} \mid y)/P(\mathbf{x}) \tag{1}$$

together with an assumption that the attributes are conditionally independent given the class. For ►attribute-value data, this assumption entitles

$$P(\mathbf{x} \mid y) = \prod_{i=1}^{n} P(x_i \mid y) \tag{2}$$

where x_i is the value of the ith attribute in $\mathbf{x}$, and n is the number of attributes.

$$P(\mathbf{x}) = \prod_{i=1}^{k} P(c_i)P(\mathbf{x} \mid c_i) \tag{3}$$

where k is the number of classes and c_i is the ith class. Thus, (1) can be calculated by normalizing the numerators of the right-hand-side of the equation.

For ►categorical attributes, the required probabilities P(y) and P($x_i \mid y$) are normally derived from frequency counts stored in arrays whose values are calculated by a single pass through the training data at training time. These arrays can be updated as new data are acquired, supporting ►incremental learning. Probability estimates are usually derived from the frequency counts using smoothing functions such as the ►Laplace estimate or an ►m-estimate.

For ►numeric attributes, either the data are discretized (see ►discretization), or probability density estimation is employed.

In ►document classification, two variants of naïve Bayes are often employed (McCallum and Nigam, 1998). The *multivariate Bernoulli model* utilizes naïve Bayes as described above, with each word in a corpus represented by a binary variable that is true if and only if the word is present in a document. However, only the words that are present in a document are considered when calculating the probabilities for that document.

In contrast, the *multinomial model* uses information about the number of times a word appears in a document. It treats each occurrence of a word in a document as a separate event. These events are assumed independent of each other. Hence the probability of a document given a class is the product of the probabilities of each word event given the class.

Cross References

►Bayes Rule
►Bayesian Methods
►Bayesian Networks
►Semi-Naïve Bayesian Learning

Recommended Reading

Lewis, D. (1998) Naive Bayes at forty: the independence assumption in information retrieval. In *Machine Learning: ECML-98, Proceedings of the 10th European Conference on Machine Learning, Chemnitz, Germany* (pp. 4–15). Berlin: Springer.

McCallum, A., & Nigam, K. (1998). A comparison of event models for Naive Bayes text classification. In *AAAI-98 Workshop on Learning for Text Categorization* (pp. 41–48). CA: AAAI Press.

NC-Learning

►Negative Correlation Learning

NCL

►Negative Correlation Learning

Nearest Neighbor

Eamonn Keogh
University California-Riverside

Synonyms

Closest point; Most similar point

Definition

In a data collection M, the *nearest neighbor* to a data object q is the data object M_i, which minimizes dist (q, M_i), where dist is a *distance measure* defined for the objects in question. Note that the fact that the object M_i is the nearest neighbor to q does not imply that q is the nearest neighbor to M_i.

Motivation and Background

Nearest neighbors are useful in many machine learning and data mining tasks, such as classification, anomaly detection, and motif discovery and in more general tasks such as spell checking, vector quantization, plagiarism detection, web search, and recommender systems.

The naive method to find the nearest neighbor to a point q requires a linear scan of all objects in M. Since this may be unacceptably slow for large datasets and/or computationally demanding distance measures, there is a huge amount of literature on speeding up nearest neighbor searches (query-by-content). The fastest methods depend on the distance measure used, whether the data is disk resident or in main memory, and the structure of the data itself. Many methods are based on the R-tree (Guttman, 1984) or one of its variants (Manolopoulos, Nanopoulos, Papadopoulos, and Theodoridis, 2005). However, in recent years there has been an increased awareness that for many applications approximate nearest neighbors may suffice. This has led to the development of techniques like *locality sensitive hashing*, which finds high-quality approximate nearest neighbors in constant time.

The definition of nearest neighbor allows for the definition of one of the simplest classification schemes, the *nearest neighbor classifier*.

The major database (SIGMOD, VLDB, and PODS) and data mining (SIGKDD, ICDM, and SDM) conferences typically feature several papers on novel distance measures and techniques for speeding up nearest neighbor search. Pavel et al.'s book provides an excellent overview on the state-of-the-art techniques in nearest neighbor searching.

Recommended Reading

Guttman, A. (1984). R-trees: A dynamic index structure for spatial searching. In *Proceedings of the 1984 ACM SIGMOD international conference on management of data* (pp. 47–57). New York: ACM. ISBN 0-89791-128-8

Manolopoulos, Y., Nanopoulos, A., Papadopoulos, A. N., & Theodoridis, Y. (2005). R-*trees: Theory and applications*. Berlin: Springer.

Zezula, P., Amato, G., Dohnal, V., & Batko, M. (2005). Similarity search: The metric space approach. In *Advances in database systems* (Vol. 32, p. 220). New York: Springer. ISBN 0-387-29146-6

Nearest Neighbor Methods

►Instance-Based Learning

Negative Correlation Learning

Synonyms

NC-learning; NCL

Definition

Negative correlation learning (Liu & Yao, 1999) is an ►ensemble learning technique. It can be used for regression or classification problems, though with classification problems the models must be capable of producing posterior probabilities. The model outputs are combined with a uniformly weighted average. The squared error is augmented with a penalty term which takes into account the diversity of the ensemble. The error for the *i*th model is,

$$E(f_i(x)) = \frac{1}{2}(f_i(x) - d)^2 - \lambda(f_i(x) - \bar{f}(x))^2. \quad (1)$$

The coefficient λ determines the balance between optimizing individual accuracy, and optimizing ensemble diversity. With $\lambda = 0$, the models are trained independently, with no emphasis on diversity. With $\lambda = 1$, the models are tightly coupled, and the ensemble is trained as a single unit. Theoretical studies (Brown, Wyatt, & Tino, 2006) have shown that NC works by directly optimizing the ►bias-variance-covariance trade-off, thus it explicitly *manages* the ensemble diversity. When the complexity of the individuals is sufficient to have high individual accuracy, NC provides little benefit. When the complexity is low, NC with a well-chosen λ can provide significant performance improvements. Thus the best situation to make use of the NC framework is with a large number of low accuracy models.

Recommended Reading

Brown, G., Wyatt, J. L., & Tino, P. (2006). Managing diversity in regression ensembles. *Journal of Machine Learning Research, 6*, 1621–1650.

Liu, Y., & Yao, X. (1999). Ensemble learning via negative correlation *Neural Networks, 12*(10), 1399–1404.

Negative Predictive Value

Negative Predictive Value (NPV) is defined as a ratio of true negatives to the total number of negatives predicted by a model. This is defined with reference to a special case of the ►confusion matrix with two classes – one designated the *positive* class and the other the *negative* class – as indicated in Table 1.

NPV can then be defined in terms of true negatives and false negatives as follows.

$$\text{NPV} = \text{TN}/(\text{TN} + \text{FN})$$

Negative Predictive Value. Table 1 The outcomes of classification into positive and negative classes

		Assigned Class	
		Positive	Negative
Actual Class	Positive	True Positive (TP)	False Negative (FN)
	Negative	False Positive (FP)	True Negative (TN)

Network Analysis

►LinkMining and Link Discovery

Network Clustering

►Graph Clustering

Networks with Kernel Functions

►Radial Basis Function Networks

Neural Networks

Neural networks are learning algorithms based on a loose analogy of how the human brain functions. Learning is achieved by adjusting the weights on the connections between nodes, which are analogous to synapses and neurons.

Cross References

►Radial Basis Function Networks

Neural Network Architecture

►Topology of a Neural Network

Neuro-Dynamic Programming

►Value Function Approximation

Neuroevolution

Risto Miikkulainen
The University of Texas at Austin
Austin, TX, USA

Synonyms

Evolving neural networks; Genetic neural networks

Definition

Neuroevolution is a method for modifying ►neural network weights, topologies, or ensembles in order to learn a specific task. Evolutionary computation (see ►Evolutionary Algorithms) is used to search for network parameters that maximize a fitness function that measures performance in the task. Compared to other neural network learning methods, neuroevolution is highly general, allowing learning without explicit targets, with non differentiable activation functions, and with recurrent networks. It can also be combined with standard neural network learning, e.g. to biological adaptation. Neuroevolution can also be seen as a policy search method for reinforcement-learning problems, where it is well suited to continuous domains and to domains where the state is only partially observable.

Motivation and Background

The primary motivation for neuroevolution is to be able to train neural networks in sequential decision tasks with sparse reinforcement information. Most neural network learning is concerned with supervised tasks, where the desired behavior is described in terms of a corpus of input to output examples. However, many learning tasks in the real world do not lend themselves to the supervised learning approach. For example, in game playing, vehicle control, and robotics, the optimal actions at each point in time are not always known; only after performing several actions, it is possible to get information about how well they worked, such as winning or losing the game. Neuroevolution makes it possible to find a neural network that optimizes behavior given only such sparse information about how well the networks are doing, without direct information about what exactly they should be doing.

The main benefit of neuroevolution compared with other reinforcement learning (RL) methods in such tasks is that it allows representing continuous state and action spaces and disambiguating hidden states naturally. Network activations are continuous, and the network generalizes well between continuous values, largely avoiding the state explosion problem that plagues many reinforcement-learning approaches. ►Recurrent networks can encode memories of past states and actions, making it possible to learn in ►partially observable Markov decision process (POMDP) environments that are difficult for many RL approaches.

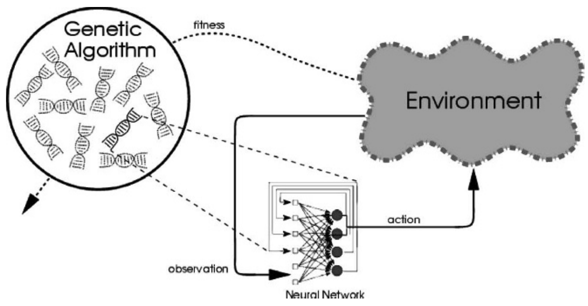
Compared to other neural network learning methods, neuroevolution is highly general. As long as the performance of the networks can be evaluated over time, and the behavior of the network can be modified through evolution, it can be applied to a wide range of network architectures, including those with non differentiable activation functions and recurrent and higher-order connections. While most neural learning algorithms focus on modifying only the weights, neuroevolution can be used to optimize other aspects of the networks as well, including activation functions and network topologies.

Third, neuroevolution allows combining evolution over a population of solutions with lifetime learning in individual solutions: the evolved networks can each learn further through, e.g., backpropagation or Hebbian learning. The approach is therefore well suited for understanding biological adaptation and building artificial life systems.

Structure of the Learning System

Basic methods

In neuroevolution, a population of genetic encodings of neural networks is evolved to find a network that solves the given task. Most neuroevolution methods follow the usual generate-and-test loop of evolutionary algorithms (Fig. 1). Each encoding in the population (a genotype) is chosen in turn and decoded into the corresponding neural network (a phenotype). This network is then employed in the task and its performance measured over time, obtaining a fitness value for the corresponding genotype. After all members of the population have been evaluated in this manner, genetic operators are used to create the next generation of the population. Those encodings with the highest fitness are mutated and crossed over with each other, and the resulting offspring replaces the genotypes with the lowest fitness in the population. The process therefore constitutes an intelligent parallel search towards better genotypes and continues until a network with a sufficiently high fitness is found.



Neuroevolution. Figure 1. Evolving neural networks. A population of genetic neural networks encodings (genotypes) is first created. At each iteration of evolution (generation), each genotype is decoded into a neural network (phenotype), which is evaluated in the task, resulting in a fitness value for the genotype. Crossover and mutation among the genotypes with the highest fitness is then used to generate the next generation

Several methods exist for evolving neural networks depending on how the networks are encoded. The most straightforward encoding, sometimes called conventional neuroevolution (CNE), is formed by concatenating the numerical values for the network weights (either binary or floating point; Floreano, Dürr, & Mattiussi, 2008; Schaffer, Whitley, & Eshelman, 1992; Yao, 1999). This encoding allows evolution to optimize the weights of a fixed neural network architecture, an approach that is easy to implement and is practical in many domains.

In more challenging domains, the CNE approach suffers from three problems. The method may cause the population to converge before a solution is found, making further progress difficult (i.e., premature convergence); similar networks, such as those where the order of nodes is different, may have different encodings and much effort is wasted in trying to optimize them in parallel (i.e., competing conventions); a large number of parameters need to be optimized at once, which is difficult through evolution.

More sophisticated encodings have been devised to alleviate these problems. One approach is to run the evolution at the level of solution components instead of full solutions. That is, instead of a population of complete neural networks, a population of network fragments, neurons, or connection weights is evolved (Gomez, Schmidhuber, & Miikkulainen, 2008; Moriarty, Schultz, & Grefenstette, 1999; Potter & Jong, 2000). Each individual is evaluated as part of a full network, and its fitness reflects how well it cooperates with other individuals in forming a full network. Specifications for how to combine the components into a full network can be evolved separately, or the combination can be based on designated roles for subpopulations. In this manner, the complex problem of finding a solution network is broken into several smaller subproblems; evolution is forced to maintain diverse solutions, and competing conventions and the number of parameters is drastically reduced.

Another approach is to evolve the network topology, in addition to the weights. The idea is that topology can have a large effect on function, and evolving appropriate topologies can achieve good performance faster than evolving weights only (Angeline, Saunders, Pollack, & An, 1994; Floreano et al., 2008; Stanley & Miikkulainen, 2004; Yao, 1999). Since topologies are explicitly specified, competing conventions are largely avoided. It is also possible to start evolution with simple solutions and gradually make them more complex, a process that takes place in biology and is a powerful approach in machine learning in general. Speciation according to the topology can be used to avoid premature convergence, and to protect novel topological solutions until their weights have been sufficiently optimized.

All of the above methods map the genetic encoding directly to the corresponding neural network, i.e., each part of the encoding corresponds to a part of the network, and vice versa. Indirect encoding, in contrast, specifies a process through which the network is constructed, such as cell division or generation through a grammar (Floreano et al., 2008; Gruau, Whitley, & Adding, 1993; Stanley & Miikkulainen, 2003; Yao, 1999). Such an encoding can be highly compact and also take advantage of modular solutions. The same structures can be repeated with minor modifications, as they often are in biology. It is, however, difficult to optimize solutions produced by indirect encoding, and realizing its full potential is still future work.

The fifth approach is to evolve an ensemble of neural networks to solve the task together, instead of a single network (Liu, Yao, & Higuchi, 2000). This approach takes advantage of the diversity in the population. Different networks learn different parts or aspects of the training data, and together the whole ensemble can perform better than a single network. Diversity can be created through speciation and negative correlation, encouraging useful specializations to emerge. The approach can be used to design ensembles for classification problems, but it can also be extended to control tasks.

Extensions

The basic mechanisms of neuroevolution can be augmented in several ways, making the process more efficient and extending it to various applications. One of the most basic ones is incremental evolution or shaping. Evolution is started on a simple task and once that is mastered, the solutions are evolved further on a more challenging task, and through a series of such transfer steps, eventually on the actual goal task itself (Gomez et al., 2008). Shaping can be done by changing the environment, such as increasing the speed of the opponents, or by changing the fitness function, e.g., by rewarding gradually more complex behaviors. It is often possible to solve challenging tasks by approaching them incrementally even when they cannot be solved directly.

Many extensions to evolutionary computation methods apply particularly well to neuroevolution. For instance, intelligent mutation techniques such as those employed in evolutionary strategies are effective because the weights often have suitable correlations (Igel, 2003). Networks can also be evolved through coevolution (Chellapilla & Fogel, 1999; Stanley & Miikkulainen, 2004). A coevolutionary arms race can be established, e.g., based on complexification of network topology: as the network becomes gradually more complex, evolution is likely to elaborate on existing behaviors instead of replacing them.

On the other hand, several extensions utilize the special properties of the neural network phenotype. For instance, neuron activation functions, initial states, and learning rules can be evolved to fit the task (Floreano et al., 2008; Yao, 1999; Schaffer et al., 1992). Most significantly, evolution can be combined with other neural network learning methods (Floreano et al., 2008). In such approaches, evolution usually provides the initial network, which then adapts further during its evaluation in the task. The adaptation can take place through Hebbian learning, thereby strengthening those existing behaviors that are invoked often during evaluation. Alternatively, supervised learning such as backpropagation can be used, provided targets are available. Even if the optimal behaviors are not known, such training can be useful. Networks can be trained to imitate the most successful individuals in the population, or part of the network can be trained in a related task such as predicting the next inputs, or evaluating the utility of actions based on values obtained through Q-learning. The weight changes may be encoded back into the genotype, implementing Lamarckian evolution; alternatively, they may affect selection through the Baldwin effect, i.e., networks that learn well will be selected for reproduction even if the weight changes themselves are not inherited (Ackley & Littman, 1992; Bryant & Miikkulainen, 2007; Gruau et al., 1993).

There are also several ways to bias and direct the learning system using human knowledge. For instance, human-coded rules can be encoded in partial network structures and incorporated into the evolving networks as structural mutations. Such knowledge can be used to implement initial behaviors in the population, or it can serve as an advice during evolution (Miikkulainen, Bryant, Cornelius, Karpov, Stanley, & Yong, 2006). In cases where rule-based knowledge is not available, it may still be possible to obtain examples of human behavior. Such examples can then be incorporated into evolution, either as components of fitness or by explicitly training the evolved solutions towards human behavior through, e.g., backpropagation (Bryant & Miikkulainen, 2007). Similarly, knowledge about the task and its components can be utilized in designing effective shaping strategies. In this manner, human expertise can be used to bootstrap and guide evolution in difficult tasks, as well as direct it towards the desired kinds of solutions.

Applications

Neuroevolution methods are powerful especially in continuous domains of reinforcement learning, and those that have partially observable states. For instance, in the benchmark task of balancing the inverted pendulum without velocity information (making the problem partially observable), the advanced methods have been shown to find solutions two orders of magnitude faster than value-function-based reinforcement-learning methods (measured by number of evaluations; Gomez et al., 2008). They can also solve harder versions of the problem, such as balancing two poles simultaneously.

The method is powerful enough to make many real-world applications of reinforcement learning possible. The most obvious area is adaptive, nonlinear control of physical devices. For instance, neural network controllers have been evolved to drive mobile robots, automobiles, and even rockets (Gomez & Miikkulainen, 2003; Nolfi & Floreano, 2000; Togelius & Lucas, 2006). The control approach have been extended to optimize systems such as chemical processes, manufacturing systems, and computer systems. A crucial limitation with current approaches is that the controllers usually need to be developed in simulation and transferred to the real system. Evolution is the strongest as an off-line learning method where it is free to explore potential solutions in parallel.

Evolution of neural networks is a natural tool for problems in artificial life. Because networks implement behaviors, it is possible to design neuroevolution experiments on how behaviors such as foraging, pursuit and evasion, hunting and herding, collaboration, and even communication may emerge in response to environmental pressure (Werner & Dyer, 1992). It is possible to analyze the evolved circuits and understand how they map to function, leading to insights into biological networks (Keinan, Sandbank, Hilgetag, Meilijson, & Ruppin, 2006). The evolutionary behavior approach is also useful for constructing characters in artificial environments, such as games and simulators. Non-player characters in current video games are usually scripted and limited; neuroevolution can be used to evolve complex behaviors for them, and even adapt them in real time (Miikkulainen et al., 2006).

Programs and Data

Software for, e.g., the NEAT method for evolving network weights and topologies, and the ESP method for evolving neurons to form networks is available at http://nn.cs.utexas.edu/keyword?neuroevolution.

The TEEM software for evolving neural networks for robotics experiments is available at http://teem.epfl.ch.

The OpenNERO software for evolving intelligent multiagent behavior in simulated environments is at http://nn.cs.utexas.edu/?opennero.

Cross References

►Evolutionary Algorithms
►Reinforcement Learning

Recommended Reading

Ackley, D., & Littman, M. (1992). Interactions between learning and evolution. In C. G. Langton, C. Taylor, J. D. Farmer, & S. Rasmussen (Eds.), *Artificial life II* (pp. 487–509). Reading, MA: Addison-Wesley.

Angeline, P. J., Saunders, G. M., & Pollack, J. B. (1994). An evolutionary algorithm that constructs recurrent neural networks. *IEEE Transactions on Neural Networks, 5*, 54–65.

Bryant, B. D., & Miikkulainen, R. (2007). Acquiring visibly intelligent behavior with example-guided neuroevolution http://nn.cs.utexas.edu/keyword?bryant:aaai07. In *Proceedings of the twenty-second national conference on artificial intelligence* (pp. 801–808). Menlo Park, CA: AAAI Press.

Chellapilla, K., & Fogel, D. B. (1999). Evolution, neural networks, games, and intelligence. *Proceedings of the IEEE, 87*, 1471–1496.

Floreano, D., Dürr, P., & Mattiussi, C. (2008). Neuroevolution: From architectures to learning. *Evolutionary Intelligence, 1*, 47–62.

Gomez, F., & Miikkulainen, R. (2003). Active guidance for a finless rocket using neuroevolution http://nn.cs.utexas.edu/keyword?gomez:gecco03. In *Proceedings of the genetic and evolutionary computation conference* (pp. 2084–2095). San Francisco: Morgan Kaufmann.

Gomez, F., Schmidhuber, J., & Miikkulainen, R. (2008). Accelerated neural evolution through cooperatively coevolved synapses http://nn.cs.utexas.edu/keyword?gomez:jmlr08. *Journal of Machine Learning Research, 9*, 937–965.

Gruau, F., & Whitley, D. (1993). Adding learning to the cellular development of neural networks: Evolution and the Baldwin effect. *Evolutionary Computation, 1*, 213–233 .

Igel, C. (2003). Neuroevolution for reinforcement learning using evolution strategies http://www.neuroinformatik.ruhr-uni-bochum.de/ini/PEOPLE/igel/NfRLUES.pdf. In R. Sarker, R. Reynolds, H. Abbass, K. C. Tan, B. McKay, D. Essam, & T. Gedeon, (Eds.), *Proceedings of the 2003 congress on evolutionary computation* (pp. 2588–2595). Piscataway, NJ: IEEE Press.

Keinan, A., Sandbank, B., Hilgetag, C. C., Meilijson, I., & Ruppin, E. (2006). Axiomatic scalable neurocontroller analysis via the Shapley value. *Artificial Life, 12*, 333–352.

Liu, Y., Yao, X., & Higuchi, T. (2000). Evolutionary ensembles with negative correlation learning. *IEEE Transactions on Evolutionary Computation, 4*, 380–387.

Miikkulainen, R., Bryant, B. D., Cornelius, R., Karpov, I. V., Stanley, K. O., & Yong, C. H. (2006). Computational intelligence in games http://nn.cs.utexas.edu/keyword?miikkulainen:cigames06. In G. Y. Yen & D. B. Fogel (Eds.), *Computational intelligence: Principles and practice* (155–191). Piscataway, NJ: IEEE Computational Intelligence Society.

Moriarty, D. E., Schultz, A. C., & Grefenstette, J. J. (1999). Evolutionary algorithms for reinforcement learning. *Journal of Artificial Intelligence Research, 11*, 199–229.

Nolfi, S., & Floreano, D. (2000). *Evolutionary robotics*. Cambridge, MA: MIT Press.

Potter, M. A., & Jong, K. A. D. (2000). Cooperative coevolution: An architecture for evolving coadapted subcomponents http://www.ncbi.nlm.nih.gov/entrez/query.fcgi?cmd=retrieve&db=pubmed&dopt=abstract&list_uids=10753229. *Evolutionary Computation, 8*, 1–29.

Schaffer, J. D., Whitley, D., & Eshelman, L. J. (1992). Combinations of genetic algorithms and neural networks: A survey of the state of the art. In D. Whitley & J. Schaffer (Eds.), *Proceedings of the international workshop on combinations of genetic algorithms and neural networks* (pp. 1–37). Los Alamitos, CA: IEEE Computer Society Press.

Stanley, K. O., & Miikkulainen, R. (2003). A taxonomy for artificial embryogeny http://nn.cs.utexas.edu/keyword?stanley:alife03. *Artificial Life, 9*, 93–130.

Stanley, K. O., & Miikkulainen, R. (2004). Competitive coevolution through evolutionary complexification http://nn.cs.utexas.edu/keyword?stanley:jair04. *Journal of Artificial Intelligence Research, 21*, 63–100.

Togelius, J., & Lucas, S. M. (2006). Evolving robust and specialized car racing skills http://algoval.essex.ac.uk/rep/games/Togelius2006Evolving.pdf. In *IEEE congress on evolutionary computation* (pp. 1187–1194). Piscataway, NJ: IEEE.

Werner, G. M., & Dyer, M. G. (1992). Evolution of communication in artificial organisms. In C. G. Langton, C. Taylor, J. D. Farmer, & S. Rasmussen (Eds.) *Proceedings of the workshop on artificial life (ALIFE '90)* (pp. 659–687). Reading, MA: Addison-Wesley.

Yao, X. (1999). Evolving artificial neural networks. *Proceedings of the IEEE, 87*(9), 1423–1447.

Neuron

Risto Miikkulainen
The University of Texas at Austin
Austin, TX, USA

Synonyms

Node; Unit

Definition

Neurons carry out the computational operations of a network; together with connections (see ►Topology of a Neural Network, ►Weights), they constitute the neural network. Computational neurons are highly abstracted from their biological counterparts. In most cases, the neuron forms a weighted sum of a large number of inputs (activations of other neurons), applies a nonlinear transfer function to that sum, and broadcasts the resulting output activation to a large number of other neurons. Such activation models the firing rate of the biological neuron, and the nonlinearity is used to limit it to a certain range (e.g., 0/1 with a threshold, (0 .. 1) with a sigmoid, (−1 .. 1) with a hyperbolic tangent, or (0 .. ∞) with an exponential function). Each neuron may also have a bias weight, i.e., a weight from a virtual neuron that is always maximally activated, which the learning algorithm can use to adjust the input sum quickly into the most effective range of the nonlinearity. Alternatively to firing rate neurons, the firing events (i.e., spikes or action potentials) of the neuron can be represented explicitly. In such an integrate-and-fire approach, each spike causes a change in the neuron's membrane potential that decays over time; an output spike is generated if the potential exceeds a threshold (see ►Biological Learning). In contrast, networks such as ►Self-Organizing Maps and ►Radial Basis Function Networks abstract the firing rate further into a measure of similarity (or distance) between the neuron's input weight vector and the vector of input activities. Learning in neural networks usually takes place by adjusting the weights on the input connections of the neuron, and can also include adjusting the parameters of the nonlinear transfer function, or the neuron's connectivity with other neurons. In this manner, the neuron converges information from other neurons, makes a simple decision based on it, broadcasts the result widely, and adapts.

Node

►Neuron

No-Free-Lunch Theorem

A theorem establishing that performance on test data cannot be deduced from performance on training data. It follows that the justification for any particular learning algorithm must be based on an assumption that nature is uniform in some way. Since different machine learning algorithms make such different assumptions, no-free-lunch theorems have been used to argue that it not possible to deduce that any algorithm is superior to any other from first principles. Thus "good" algorithms are those whose ►inductive bias matches the way the world happens to be.

Nogood Learning

Nogood learning is a ►deductive learning technique used for the purpose of ►intelligent backtracking in constraint satisfaction. The approach analyzes failures at backtracking points and derives sets of variable bindings, or *nogoods*, that will never lead to a solution. These nogood constraints can then be used to prune later search nodes.

Noise

The training data for a learning algorithm is said to be *noisy* if the data contain errors. Errors can be of two types:

- A *measurement error* occurs when some attribute values are incorrect or inaccurate. Note that measurement of physical properties by continuous values is always subject to some error.
- In supervised learning, *classification error* means that a training example has an incorrect class label.

In addition to errors, training examples may have ►missing attribute values. That is, the values of some attribute values are not recorded.

Noisy data can cause learning algorithms to fail to converge to a concept description or to build a concept description that has poor classification accuracy on unseen examples. This is often due to ►over fitting.

For methods to minimize the effects of noise, see ►Over Fitting.

Nominal Attribute

A **nominal attribute** assumes values that classify data into mutually exclusive (nonoverlapping), exhaustive, unordered categories. See ►Attribute and ►Measurement Scales.

Nonparametric Bayesian

►Gaussian Process

Nonparametric Cluster Analysis

►Density-Based Clustering

Non-Parametric Methods

►Instance-Based Learning

Nonstandard Criteria in Evolutionary Learning

Michele Sebag
Université Paris-Sud, Orsay, France

Introduction

Machine learning (ML), primarily concerned with extracting models or hypotheses from data, comes into three main flavors: ►supervised learning also known as ►classification or ►regression (Bishop, 2006; Duda et al., 2001; Han and Kamber, 2000), ►unsupervised learning also known as ►clustering (Ben-David et al., 2005), and ►reinforcement learning (Sutton and Barto, 1998).

All three types of problems can be viewed as optimization problems. The ML core task is to define a *learning criterion* (i.e., the function to be optimized) such that it enforces (i) the statistical relevance of the solution; (ii) the well-posedness of the underlying optimization problem. Since evolutionary computation (see ►Evolutionary Algorithms) makes it possible to handle ill-posed optimization problems, the field of evolutionary learning (Holland, 1986) has investigated quite a few nonstandard learning criteria and search spaces. Only supervised ML will be considered in the following. Unsupervised learning has hardly been touched upon in the evolutionary computation (EC) literature; regarding reinforcement learning, the interested reader is referred to the entries related to ►evolutionary robotics and control.

The entry will first briefly summarize the formal background of supervised ML and its two mainstream approaches for the last decade, namely support vector machines (SVMs) (Cristianini and Shawe-Taylor, 2000; Schölkopf et al., 1998; Vapnik, 1995) and ensemble learning (Breiman 1998; Dietterich, 2000; Schapire, 1990). Thereafter and without pretending to exhaustivity, this entry will illustrate some innovative variants of these approaches in the literature, building upon the evolutionary freedom of setting and tackling optimization problems.

Formal Background

Supervised learning exploits a dataset $\mathcal{E} = \{(\mathbf{x}_i, y_i), \mathbf{x}_i \in X, y_i \in Y, i = 1 \ldots n\}$, where X stands for the instance space (e.g., $\mathbb{R}^d$), Y is the label space, and $(\mathbf{x}_i, y_i)$ is a labeled example, as depicted in Table 1. Supervised learning is referred to as *classification* (respectively *regression*) when Y is a finite set (respectively when $Y = \mathbb{R}$).

The ML goal is to find a hypothesis or classifier $h : X \mapsto Y$ such that $h(x)$ is "sufficiently close" to the true label y of x for any x ranging in the instance domain. It is generally assumed that the available examples are independently and identically distributed (iid) after a probability distribution P_{XY} on $X \times Y$. Letting $\ell(y', y)$ denote the loss incurred by labeling $\mathbf{x}$ as y' instead of its true label y, the learning criterion is most naturally defined as the expectation of the loss, or *generalization*

Nonstandard Criteria in Evolutionary Learning. Table 1 Excerpt of a dataset in a failure identification problem (binary classification). Instance space *X* is the cross product of all attribute domains: for example, attribute *Temperature* ranges in R, attribute *Material* ranges in $\{Ni, Fe, \ldots\}$. Label space *Y* is binary

	Temperature	Material	Aging	Label
$\mathbf{x}_1$	118.2	Ni	No	Failure
$\mathbf{x}_2$	76.453	Fe	Yes	OK

error, to be minimized, where $\mathcal{H}$ denotes the hypothesis space:

$$\text{Find } h^* = arg\min\{\mathcal{F}(h) = \int \ell(h(x), y)dP(x, y),\ h \in \mathcal{H}\}$$

The generalization error however is not computable, since the joint distribution P_{XY} of instances and labels is unknown; only its approximation on the training set, referred to as *empirical error*, can be computed as follows:

$$\mathcal{F}_e(h) = \frac{1}{n}\sum_{i=1}^{n} \ell(h(\mathbf{x}_i), y_i)$$

Using results from the theory of measure and integration, the generalization error is upper bounded by the empirical error, plus a term reflecting the number of examples and the regularity of the hypothesis class (Fig. 1).

Note that minimizing the empirical error alone leads to the infamous *overfitting* problem: while the predictive accuracy on the training set is excellent, the error on a (disjoint) test set is much higher. All learning criteria thus involve a trade-off between the empirical error and a so-called regularization term, providing good guarantees (upper bound) on the generalization error.

In practice, learning algorithms also involve hyperparameters (e.g., the weight of the regularization term). These are adjusted using cross-validation using a grid search (EC approaches have also been used to find optimal learning hyperparameters, ranging from the topology of neural nets [Miikkulainen et al., 2003], to the kernel parameters in SVM [Friedrichs and Igel, 2005; Mierswa, 2006].) The dataset is divided into K subsets with same class distribution; hypothesis h_i is learned from the training set made of all subsets except the i-th and the empirical error of h_i is measured on the ith subset. An approximation of the generalization error is provided by the average of the h_i errors when $i = 1 \ldots K$, referred to as cross-fold error, and the hyperparameter setting is empirically determined to minimize the cross-fold error.

Support Vector Machines

Considering a real-valued instance space ($X = \mathbb{R}^D$), a linear ►support vector machine (SVM) (Boser et al., 1992) constructs the separating hyperplane (where $< a, b >$ stands for the dot product of vectors a and b):

$$h(\mathbf{x}) = \ < w, \mathbf{x} > +b$$

which maximizes the margin that is, the minimal distance between the examples and the hyperplane, when such separating hyperplanes exists (Fig. 2). A slightly more complex formulation, involving the so-called slack variables $\mathbf{x}i_i$, is defined to deal with noise (Cortes and Vapnik, 1995).

The function to be optimized, the L_2 norm of the hyperplane normal vector w, is quadratic; using Lagrange multipliers to account for the constraints gives rise to the so-called dual formulation. Let us call *support vectors* those examples for which the constraint is active (Lagrange multiplier $\alpha_i > 0$), then it becomes

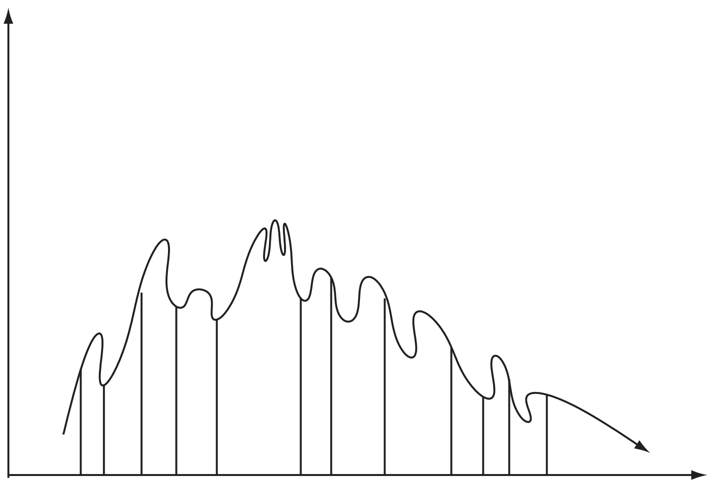
$$h(\mathbf{x}) = \sum y_i\alpha_i < \mathbf{x}_i, \mathbf{x} > +b \quad \text{with } \alpha_i > 0; \quad \sum \alpha_i y_i = 0$$

As will be seen in section "Evolutionary Regularization," this formulation defines a search space, which can be directly explored by EC (Mierswa, 2007).

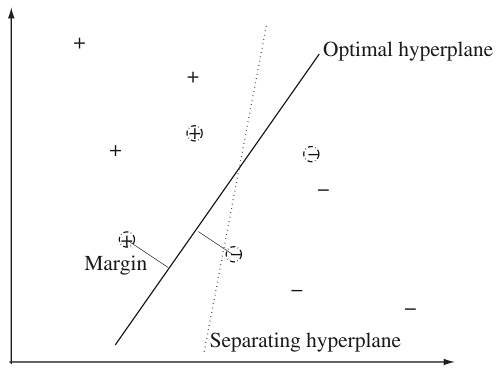
Obviously however, linear classifiers are limited. The power of SVMs comes from the so-called kernel *trick*, naturally exporting the SVM approach to nonlinear hypothesis spaces. Let us map the instance space X onto some *feature space* X' via mapping Φ. If the scalar product on X' can be computed in X (e.g., $< \Phi(\mathbf{x}), \Phi(\mathbf{x}') >=_{def} K(x, x')$) then a linear classifier in X' (nonlinear with reference to X) is given as

For an iid. sample $\mathbf{x}_1, \ldots \mathbf{x}_n$, for $g \in \mathcal{G}$

$$\int g(x)dx < \frac{1}{n}\sum_{i=1}^{n} g(\mathbf{x}_i) + C(n, \mathcal{G})$$



Nonstandard Criteria in Evolutionary Learning. Figure 1. Bounding the integral from the empirical average depending on the uniform sample size and the class of functions $\mathcal{G}$ at hand



without noise

$$\text{Minimize } \frac{1}{2}||w||^2$$

$$\text{s.t. for } i = 1 \text{ to } n$$

$$y_i(< w, \mathbf{x}_i > +b) \geq 1$$

with noise

$$\text{Minimize } \frac{1}{2}||w||^2 + C\sum_{i=1}^{n} \mathbf{x}i_i$$

$$\text{s.t. for } i = 1 \text{ to } n$$

$$y_i(< w, \mathbf{x}_i > +b) \geq 1 - \mathbf{x}i_i; \quad \mathbf{x}i_i \geq 0$$

Nonstandard Criteria in Evolutionary Learning. Figure 2. Linear support vector machines. The optimal hyperplane is the one maximizing the minimal distance to the examples

$h(\mathbf{x}) = \sum_i y_i \alpha_i K(\mathbf{x}_i, \mathbf{x}) + b$. The only requirement is to use a positive definite kernel (ensuring that the underlying optimization problem is well posed). Again, this requirement can be relaxed in the evolutionary learning framework (Mierswa, 2006).

Among the most widely used kernels are the Gaussian kernel $\left(K(x, x') = exp\left\{-\frac{||x-x'||^2}{\sigma^2}\right\}\right)$ and the polynomial kernel $(K(x, x') = (<x, x'> + c)^d)$. The kernel parameters σ, c, d, referred to as learning hyper-parameters, have been tuned by some authors using EC, as well as the kernel itself (see among others (Friedrichs and Igel, 2005; Gagné et al., 2006; Mierswa, 2006)).

Ensemble methods

The other mainstream approach in supervised learning, ►ensemble learning (EL), relies on somewhat different principles. Schapire's seminal paper, *The strength of weak learnability*, exploring the relationship between *weak learnability* (ability of building a hypothesis slightly better than random guessing, whatever the distribution of the dataset is (C)) and *strong learnability* (ability of building a hypothesis with arbitrarily high predictive accuracy), established a major and counterintuitive result: strong and weak learnability are equivalent (Schapire, 1990). The idea behind the proof is that combining many weak hypotheses learned under different distributions yields an arbitrarily accurate hypothesis. As the errors of the weak hypotheses should not concentrate in any particular region of the instance space (for condition C to hold), the law of large numbers states that averaging them leads to exponentially decrease the empirical error.

Two main EL approaches have been investigated in the literature. The first one, ▶bagging (Breiman, 1998), builds a large number of independent hypotheses; the source of variations is bootstrapping (uniformly selecting the training set with replacement from the initial dataset); or varying the parameters of the learning algorithm; or subsampling the features considered at each step of the learning process (Amit et al., 1997; Breiman, 2001). The final classifier is usually obtained by averaging these solutions.

The other EL approach, ▶boosting (Freund and Shapire, 1996), iteratively builds a sequence of hypotheses, where each h_i somehow is in charge of correcting the mistakes of $h_1, \ldots h_{i-1}$. Specifically, a distribution $\mathcal{W}_t$ is defined on the training set at step t, with $\mathcal{W}_0$ being the uniform distribution. At step t, the weight of every example misclassified by h_t is increased (multiplied by $exp\{-h_t(\mathbf{x}_i).h_i\}$; then a normalization step follows to ensure that $\mathcal{W}_{t+1}$ still sums to 1); hypothesis h_{t+1} will thus focus on the examples misclassified by h_t. Finally, the classifier is defined as the weighted vote of all h_t.

The intuition behind boosting is that not all examples are equal: some examples are more difficult than others (more hypotheses misclassify them) and the learning process should thus focus on these examples (with the caveat that a difficult example might be so because it is noisy). Interestingly, the intuition that examples are not equal has been formalized in terms of coevolution (When designing a program, the fitness of the candidate solutions is computed after some test cases; for the sake of accuracy and feasability, the difficulty and number of test cases must be commensurate with the competence of the current candidate solutions. Hillis defined a competitive coevolution setting between the program species and the test case species: while programs aim at solving test cases, test cases aim at defeating candidate programs. This major line of research however is outside the scope of evolutionary learning as it assumes that the whole distribution P_{XY} is known.) by D. Hillis in the early 1990s (Hillis, 1990).

Many empirical studies suggest that boosting is more effective than bagging (with some caveat in the case of noisy domains), thanks to the higher diversity of the boosting ensemble (Dietterich, 2000; Margineantu and Dietterich, 1997).

In the ensemble learning framework, the *margin* of an example $\mathbf{x}$ is defined as the difference between the (cumulated weight or number) of hypotheses labeling $\mathbf{x}$ as positive, and those labeling $\mathbf{x}$ as negative. Like in the SVM framework, the margin of an example reflects the confidence of its classification (how much this example should be perturbed for its label to be modified).

Learning Criteria

Learning criterion and *fitness function* will be used interchangeably in the following. Since Holland's seminal papers on evolutionary learning (Holland, 1986, 1975), the most used learning criterion is the predictive accuracy on the available dataset. After the early 1990s however, drawbacks related to either learning or evolutionary issues motivated the design of new fitness functions.

Evolutionary Regularization

In the ▶genetic programming field, the early use of more sophisticated learning criteria was motivated by the so-called bloat phenomenon (Banzhaf and Langdon, 2002; Poli, 2008), that is, the uncontrolled growth of the solution size as evolution goes on. Two main approaches have been considered. The first one boils down to regularization (section "Formal Background"): the fitness function is composed of the predictive accuracy plus an additional term meant to penalize large-sized solutions (Blickle, 1996). The tricky issue of course is how to adjust the weight of the penalization term; the statistical ML theory offers no principled solution to this issue (except in an asymptotic perspective, when the number of training examples goes to infinity (Gelly et al., 2006)); thus, the weight is adjusted empirically using cross-validation (section "Formal Background").

Another approach (Blickle, 1996) is based on the use of two fitness functions during the same evolution run, after the so-called behavioral memory paradigm (Schoenauer and Xanthakis, 1993). In a first phase, the population is evolved to maximize the predictive accuracy. In a second phase, the optimization goal becomes to minimize the solution size *while preserving the predictive accuracy* reached in the former phase. As could have been expected, this second approach also depends upon the careful empirical adjustment of hyper-parameters (when to switch from one phase to another one).

Another approach is to consider regularized learning as a multi-objective optimization problem, avoiding the computationally heavy tuning of the regularization weight (Note however that in the case where the regularization involves the L_1 norm of the solution, the Pareto front can be analytically derived using the celebrated LASSO algorithm (Hastie et al., 2004; Tibshirani, 1996).). Mierswa (2007) applies multi-objective evolutionary optimization, specifically NSGA-II ([Deb et al., 2000]; see the Multi-Objective Evolutionary Optimization entry in this encyclopedia), to the simultaneous optimization of the margin and the error. The search space is nicely and elegantly derived from the dual form of SVMs (section "Support Vector Machines"): it consists of vectors $(\alpha_1, \ldots \alpha_n)$, where most α_i are zero and $\sum_i \alpha_i y_i = 0$. A customized mutation operator, similar in spirit to the sequential minimization optimization proposed by Platt [1999], enables to explore the solutions with few support vectors. The Pareto front shows the trade-off between the regularization term and the training error. At some point however, a hold-out (test set) needs be used to detect and avoid overfitting solutions, boiling down to cross-validation. Another multi-objective optimization learning is proposed by Suttorp and Igel (2006) (see section "AUC Area Under the Roc Curve").

Ensemble Learning and Boosting

Ensemble learning and evolutionary computation share two main original features. Firstly, both rely on a population of candidate solutions; secondly, the diversity of these solutions commands the effectiveness of the approach. It is no surprise therefore that evolutionary ensemble learning, tightly coupling EC and EL, has been intensively investigated in the last decade (Another exploitation of the hypotheses built along independent evolutionary learning runs concerns feature selection (Jong et al., 2004), which is outside the scope of this entry.)

A family of diversity-oriented learning criteria has been investigated by Xin Yao and collaborators, switching the optimization goal from "learning the best hypothesis" toward "learning the best ensemble" (Monirul Islam and Yao, 2008). The hypothesis space is that of neural networks (NNs). Nonparametric and parametric operators are used to simultaneously optimize the neural topology and the NN weights. Among parametric operators is the gradient-based backpropagation (BP) algorithm to locally optimize the weights (Rumelhart and McClelland, 1986), combined with simulated annealing to escape BP local minima.

Liu et al. (2000) enforce the diversity of the networks using a *negative correlation* learning criterion. Specifically, the BP algorithm is modified by replacing the error of the t-th NN on the i-th example with a weighted sum of this error and the error of the ensemble of the other NNs; denoting H_{-t} the ensemble made of all NNs but the tth one:

$$(h_t(\mathbf{x}_i) - y_i)^2 \rightarrow (1-\lambda)(h_t(\mathbf{x}_i) - y_i)^2 + \lambda(H_{-t}(\mathbf{x}_i) - y_i)^2)$$

Moreover, ensemble negative correlation–based learning exploits the fact that not all examples are equal, along the same line as boosting (section "Ensemble Methods"): to each training example is attached a weight, reflecting the number of hypotheses that misclassify it; finally the fitness associated to each network is the sum of the weights of all examples it correctly classifies. While this approach nicely suggests that ensemble learning is a multiple objective optimization (MOO) problem (minimize the error rate and maximize the diversity), it classically handles the MOO problem as a fixed weighted sum of the objectives (the value of parameter λ is fixed by the user).

The MOO perspective is further investigated by Chandra and Yao in the DIVACE system, enforcing the multilevel evolution of ensemble of classifiers (Chandra and Yao, 2006a,b). In (Chandra and Yao, 2006b), the top-level evolution simultaneously minimizes the error rate (accuracy) and maximizes the negative correlation (diversity). In (Chandra and Yao, 2006a), the negative correlation-inspired criterion is replaced by a *pairwise failure crediting*; the difference concerns the misclassification of examples that are correctly classified by other classifiers. Several heuristics have been investigated to construct the ensemble from the last population, based on averaging the hypothesis values, using the (weighted) vote of all hypotheses, or selecting a subset of hypotheses, for example, by clustering the final hypothesis population after their phenotypic distance, and selecting a hypothesis in each cluster.

Gagné et al. (2007) tackle both the construction of a portfolio of classifiers, and the selection of a subset thereof, either from the final population only as in (Chandra and Yao, 2006a,b), or from all generations. In order to do so, a reference set of classifiers is used to define a dynamic optimization problem: the fitness of a candidate hypothesis reflects whether h improves on the reference set; in the meantime, the reference set is updated every generation. Specifically, noting w_i the fraction of reference classifiers misclassifying the i-th example, $\mathcal{F}(h)$ is set to the sum of w_i^γ, taken over all examples correctly classified by h. Parameter γ is used to mitigate the influence of noisy examples.

Boosting and Large-Scale Learning

Another key motivation for designing new learning criteria is to yield scalable learning algorithms, coping with giga or terabytes of data (see [Sonnenburg et al., 2008]).

Song et al. (2003, 2005) presented an elegant genetic programming approach to tackle the intrusion detection challenge (Lippmann et al., 2000); this challenge offers a 500,000 pattern training set, exceeding standard available RAM capacities. The proposed approach relies on the dynamic subset selection method first presented by Gathercole and Ross (1994). The whole dataset is equally and randomly divided into subsets $\mathcal{E}_i$ with same distribution as the whole dataset, where each $\mathcal{E}_i$ fits within the available RAM. Iteratively, some subset $\mathcal{E}_i$ is selected with uniform probability, and loaded in memory; it is used for a number of generations set to $G_{max} \times Err(i)$ where G_{max} is the user-supplied maximum number of generations, and $Err(i)$ is the minimum number of patterns in $\mathcal{E}_i$ misclassified the previous time $\mathcal{E}_i$ was considered. Within $\mathcal{E}_i$, a competition is initiated between training patterns to yield a frugal yet challenging assessment of the hypotheses. Specifically, every generation or so, a restricted subset is selected by tournament in $\mathcal{E}_i$, considering both the difficulty of the patterns (the difficulty of pattern $\mathbf{x}_j$ being the number of hypotheses misclassifying $\mathbf{x}_j$ last time $\mathbf{x}_j$ was selected) and its age (the number of generations since $\mathbf{x}_j$ was last selected). With some probability (30% in the experiments), the tournament returns the pattern with maximum age; otherwise, it returns the pattern with maximum difficulty.

The dynamic selection subset (DSS) heuristics can thus be viewed as a mixture of uniform sampling (modeled by the age-based selection) and boosting (corresponding to the difficulty-based selection). This mixed distribution gets the best of both worlds: it speeds up learning by putting the stress on the most challenging patterns, akin boosting; in the meanwhile, it prevents noisy examples from leading learning astray as the training set always includes a sufficient proportion of uniformly selected examples. The authors report that the approach yields accurate classifiers (though outperformed by the Challenge winning entry), while one trial takes 15 min on a modest laptop computer (1 GHz Pentium, 256 MB RAM).

Gagné et al., aiming at the scalable optimization of SVM kernels, proposed another use of dynamic selection subset in a coevolutionary perspective (Gagné et al., 2006). Specifically, any kernel induces a similarity on the training set

$$s(\mathbf{x}, \mathbf{x}') = 2K(\mathbf{x}, \mathbf{x}') - K(\mathbf{x}, \mathbf{x}) - K(\mathbf{x}', \mathbf{x}')$$

This similarity directly enables the classification of examples along the k-nearest neighbor approach (Duda et al., 2001) (see ►Nearest Neighbor), labeling an example after the majority of its neighbors. Inspired from (Gilad-Bachrach et al., 2004), the margin of an example is defined as the rank of its closest neighbor in the same class, minus the rank of its closest neighbor in the other class (the closer a neighbor, the higher its rank is). The larger the margin of an example, the more confident one can be it will be correctly classified; the fitness of the kernel could thus be defined as the sum of the example margins. Computed naively however, this fitness would be quadratic in the size of the training set, hindering the scalability of the approach.

A three-species coevolutionary framework was thus defined. The first species is that of kernels; the second species includes the candidate neighbor instances, referred to as prototypes; the third species includes the training instances, referred to as test cases. Kernels and prototypes undergo a cooperative co-evolution: they cooperate to yield the underlying metric (similarity) and the reference points (prototypes) enabling to classify all training instances. The test cases, in the meanwhile, undergo a competitive coevolution with the other two species: they present the learning process with more and more difficult training examples, aiming at a good coverage of the whole instance space. The approach reportedly yields accurate kernels at a moderate computational cost.

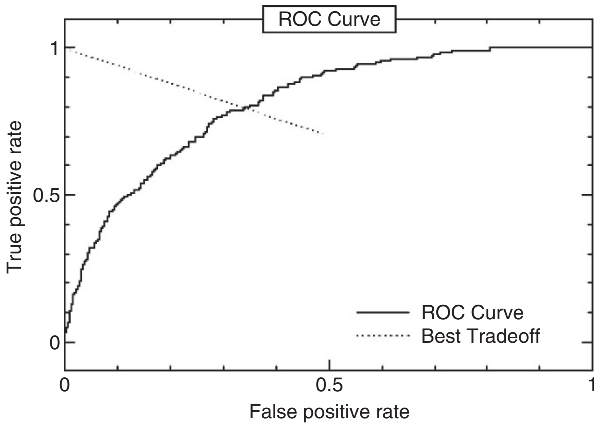
AUC: Area Under the ROC Curve

The misclassification rate criterion is notably ill-suited to problem domains with a minority class. If the goal is to discriminate a rare disease (< 1% of the training set) from a healthy state, the default hypothesis ("everyone is healthy" with 1% misclassified examples) can hardly be outperformed in terms of predictive accuracy. Standard heuristics accommodating ill-balanced problems involve the oversampling of the minority class, undersampling of the majority class, or cost-sensitive loss function (e.g., misclassifying a healthy person for an ill one costs 1, whereas the opposite costs 100) (Domingos, 1999).

Another principled approach is based on the so-called area under the receiver-operating characteristics curve (see ▶ROC Analysis). Let us consider a continuous hypothesis h, mapping the instance space on the real-value space $\mathbb{R}$. For each threshold τ let the binary classifier h_τ be defined as instance x is positive iff $h(x) > \tau$. To each τ value can be associated the true positive (TP) rate (fraction of ill persons that are correctly classified) and the false positive (FP) rate (fraction of healty persons misclassified as ill ones). In the (FP,TP) plane, the curve drawn as τ varies defines the ROC curve (Fig. 3).

Noting that the ideal classifier lies in the upper left corner (0% false positive rate, 100% true positive rate), it comes naturally to optimize the area under the ROC curve. This criterion, also referred to as Wilcoxon rank test, has been intensively studied in both theoretical and algorithmic perspectives (see among many others (Cortes and Mohri, 2004; Ferri et al., 2002; Joachims, 2005; Rosset, 2004)).

The AUC criterion has been investigated in the EC literature since the 1990s (Fogel et al., 1998), for it defines a combinatorial optimization problem. Considering the search space of real-valued functions, mapping instance



Nonstandard Criteria in Evolutionary Learning. Figure 3. The receiver operating characteristic (ROC) Curve depicts how the true positive (TP) rate increases vs the false positive (FP) rate. Random guessing corresponds to the diagonal line. The ROC curve is insensitive to ill-balanced distributions as TP and FP rates are normalized

space X onto $\mathbb{R}$, the AUC (Wilcoxon) criterion is defined as

$$\mathcal{F}(h) = \; Pr(h(\mathbf{x}) > h(\mathbf{x}')|y > y')$$
$$\mathcal{F}_e(h) \propto \; \#\{(\mathbf{x}_i, \mathbf{x}_j) \; s.t. \; h(\mathbf{x}_i) > h(\mathbf{x}_j), y_i = 1, y_j = 0\}$$

Specifically, hypothesis h is used to rank the instances; any ranking such that all positive instances are ranked before the negative ones gets the optimal AUC. The fitness criterion can be computed with complexity $\mathcal{O}(n \log n)$ where n stands for the number of training instances, by showing that

$$\mathcal{F}_e(h) \propto \sum_{i=1\ldots n, y_i=1} i \times rank(i)$$

Interestingly, the optimization of the AUC criterion can be dealt with in the SVM framework, as shown by Joachims (2005), replacing class constraints by inegality constraints (Fig. 2):

$$y_i(< w, \mathbf{x}_i > +b) \geq 1 \quad i = 1 \ldots n$$
$$\rightarrow \quad < w, \mathbf{x}_i - \mathbf{x}_j > \;\; \geq 1 \quad i, j = 1 \ldots n, \; s.t. \; y_i > y_j$$

In practice, the quadratic optimization process introduces gradually the violated constraints only, to avoid dealing with a quadratic number of constraints.

The flexibility of EC can still allow for more specific and application-driven interpretation of the AUC criterion. Typically in medical applications, the physician is most interested in the beginning of the AUC curve, trying to find a threshold τ retrieving a high fraction of ill patients for a very low false positive rate. The same situation occurs in customer relationship management, replacing positive cases by potential churners. The AUC criterion can be easily adapted to minimize the number of false positive within the top k-ranked individuals, as shown by Mozer et al., (2001).

In a statistical perspective however (and contrarily to a common practice in the ML and data mining communities), it has been argued that selecting a classifier based on its AUC was not appropriate (David J. Hand, 2009). The objection is that the AUC maximization yields the best hypothesis *under a uniform distribution of the misclassification costs*, whereas hypothesis h is used with a specific threshold τ, corresponding to a particular point of the ROC curve (Fig. 3).

Still, ROC curves convey very clear intuitions about the trade-off between TP and FP rates; analogous to a Pareto front, they enable one to select a posteriori the best trade-off according to a one's implicit preferences. An interesting approach along these lines has been investigated by Suttorp and Igel (2006) to learn SVMs, using a multi-objective optimization setting to simultaneously minimize the FP rate, and maximize the TP rate, and maximize the number of support vectors.

The last objective actually corresponds to a regularization term: the empirical error plus the number of support vectors upper-bounds the so-called leave-one-out error (when the number of folds in cross-fold validation is set to the number of examples), since the hypothesis is not modified when removing a non-support vectors. (see Zhang [2003] for more detail).

Conclusions

Unsurprisingly, the bottom line of evolutionary learning matches that of EC: any effort to customize the fitness function is highly rewarded; a good knowledge of the domain application enables to choose appropriate, frugal yet effective, search space and variation operators.

Another message concerns the validation of the proposed approaches. In early decades, hypotheses were assessed from their training error, with poor applicative relevance due to overfitting. Better practices are now widely used (e.g., training, validation, and test sets); as advocated by Dietterich (1998), good practices are based on cross-validation. Taking into account early remarks about the University of California Irvine (UCI) repository (Holte, 1993), experimental validation should consider actually challenging problems.

Due to space limitations, this entry has excluded some nice and elegant work at the crossroad of machine learning and evolutionary computation, among others, interactive optimization and modelisation of the user's preferences (Llorà et al., 2005), interactive feature construction (Krawiec and Bhanu, 2007; Venturini et al., 1997), or ML-based heuristics for noisy optimization (Heidrich-Meisner and Igel, 2009).

Recommended Reading

Amit, Y., Geman, D., & Wilder, K. (1997). Joint induction of shape features and tree classifiers. *IEEE Transactions on Pattern Analysis and Machine Intelligence, 19*(11), 1300–1305.

Banzhaf, W., & Langdon, W. B. (2002). Some considerations on the reason for bloat. *Genetic Programming and Evolvable Machines, 3*(1), 81–91.

Ben-David, S., von Luxburg, U., Shawe-Taylor, J., & Tishby, N. (Eds.). (2005). *Theoretical foundations of clustering*. NIPS Workshop.

Bishop, C. (2006). *Pattern recognition and machine learning*. Springer.

Blickle, T. (1996). Evolving compact solutions in genetic programming: a case study. In H.-M. Voigt et al. (Eds.), *Proceedings of the 4th international inference on parallel problem solving from nature. Lecture notes in computer science* (vol. 1141, pp. 564–573). Berlin: Springer.

Boser, B., Guyon, I., & Vapnik, V. (1992). A training algorithm for optimal margin classifiers. *Proceedings of the 5th annual ACM conference on Computational Learning Theory, COLT'92*, (pp. 144–152). Pittsburgh, PA.

Breiman, L. (1998). Arcing classifiers. *Annals of Statistics, 26*(3), 801–845.

Breiman, L. (2001). Random forests. *Machine Learning, 45*(1), 5–32.

Chandra, A., & Yao, X. (2006). Ensemble learning using multi-objective evolutionary algorithms. *Journal of Mathematical Modelling and Algorithms, 5*(4), 417–425.

Chandra, A., & Yao X. (2006). Evolving hybrid ensembles of learning machines for better generalisation. *Neurocomputing, 69*, 686–700.

Cortes, C., & Vapnik, V. N. (1995). Support-vector networks. *Machine Learning, 20*, 273–297.

Cortes, C., & Mohri, M. (2004). Confidence intervals for the area under the ROC curve. *Advances in Neural Information Processing Systems, NIPS*, 17.

Cristianini, N., & Shawe-Taylor, J. (2000). *An introduction to support vector machines and other kernel-based learning methods*. Cambridge: Cambridge University Press.

David J. Hand. (2009). Measuring classifier performance: a coherent alternative to the area under the ROC curve. *Machine Learning*, *77*(1), 103–123. http://dx.doi.org/10.1007/S10994-009-5119-5, DBLP, http://dblp.uni-trier.de

Deb, K., Agrawal, S., Pratab, A., & Meyarivan, T. (2000). A fast elitist non-dominated sorting genetic algorithm for multi-objective optimization: NSGA-II. In M. Schoenauer et al. (Eds.), *Proceedings of the parallel problem solving from nature VI conference*, Paris, France, pp. 849–858. Springer. Lecture Notes in Computer Science No. 1917.

Dietterich, T. G. (1998). Approximate statistical tests for comparing supervised classification learning algorithms. *Neural Computation, 10*, 1895–1923.

Dietterich, T. (2000). Ensemble methods in machine learning. In J. Kittler & F. Roli (Eds.), *First International Workshop on Multiple Classifier Systems*, Springer, pp. 1–15.

Domingos, P. (1999). Meta-cost: a general method for making classifiers cost sensitive. In *Proceedings of the 5th ACM SIGKDD International Conference on Knowledge discovery and data mining*, (pp. 155–164). San Diego, CA: ACM.

Duda, R. O., Hart, P. E., & Stork, D. G. (2001). *Pattern classification* (2nd ed.). New York: Wiley.

Ferri, C., Flach, P. A., & Hernndez-Orallo, J. (2002). Learning decision trees using the area under the ROC curve. In C. Sammut & A. G. Hoffman (Eds.), *Proceedings of the Nineteenth International Conference on Machine Learning (ICML 2002)*, (pp. 179–186). Morgan Kaufmann.

Fogel, D. B., Wasson, E. C., Boughton, E. M., Porto, V. W., and Angeline, P. J. (1998). Linear and neural models for classifying breast cancer. *IEEE Transactions on Medical Imaging, 17*(3), 485–488.

Freund, Y., & Shapire, R. E. (1996). Experiments with a new boosting algorithm. In L. Saitta (Ed.), *Proceedings of the Thirteenth International Conference on Machine Learning (ICML 1996)*, (pp. 148–156). Bari: Morgan Kaufmann.

Friedrichs, F., & Igel, C. (2005). Evolutionary tuning of multiple SVM parameters. *Neurocomputing, 64*(C), 107–117.

Gagné, C., Schoenauer, M., Sebag, M., & Tomassini, M. (2006). Genetic programming for kernel-based learning with co-evolving subsets selection. In T. P. Runarsson, H.-G. Beyer, E. K. Burke, J. J. Merelo Guervós, L. Darrell Whitley, & X. Yao (Eds.), *Parallel problem solving from nature – PPSN IX, volume 4193 of Lecture Notes in Computer Science* (pp. 1008–1017). Springer.

Gagné, C., Sebag, M., Schoenauer, M., & Tomassini, M. (2007). Ensemble learning for free with evolutionary algorithms? In H. Lipson (Ed.), *Genetic and Evolutionary Computation Conference, GECCO 2007*, (pp. 1782–1789). ACM.

Gathercole, C., & Ross, P. (1994). Dynamic training subset selection for supervised learning in genetic programming. In *Parallel problem solving from nature – PPSN III, volume 866 of lecture notes in computer science* (pp. 312–321). Springer.

Gelly, S., Teytaud, O., Bredeche, N., & Schoenauer, M. (2006). Universal consistency and bloat in GP: Some theoretical considerations about genetic programming from a statistical learning theory viewpoint. *Revue d'Intelligence Artificielle, 20*(6), 805–827.

Gilad-Bachrach, R., Navot, A., & Tishby, N. (2004). Margin based feature selection – theory and algorithms. *Proceedings of the Twenty-First International Conference on Machine Learning (ICML 2009)*, ACM Press, p. 43.

Han, J., & Kamber, M. (2000). *Data mining: concepts and techniques*. New York: Morgan Kaufmann.

Hastie, T., Rosset, S., Tibshirani, R., & Zhu, J. (2004). The entire regularization path for the support vector machine. *Advances in Neural Information Processing Systems, NIPS* 17.

Heidrich-Meisner, V., & Igel, C. (2009). Hoeffding and Bernstein races for selecting policies in evolutionary direct policy search. *Proceedings of the Twenty-Sixth International Conference on Machine Learning (ICML 2009)*, ACM, pp. 401–408.

Hillis, W. D. (1990). Co-evolving parasites improve simulated evolution as an optimization procedure. *Physica D, 42*, 228–234.

Holland, J. (1986). Escaping brittleness: The possibilities of general purpose learning algorithms applied to parallel rule-based systems. In R. S. Michalski, J. G. Carbonell, & T. M. Mitchell (Eds.), *Machine learning: an artificial intelligence approach* (vol. 2, pp. 593–623). Morgan Kaufmann.

Holland, J. H. (1975). *Adaptation in natural and artificial systems*. Ann Arbor: University of Michigan Press.

Holte, R. C. (1993). Very simple classification rules perform well on most commonly used datasets. *Machine Learning, 11*, 63–90.

Monirul Islam, M., & Yao, X. Evolving artificial neural network ensembles. In J. Fulcher & L. C. Jain (Eds.), *Computational intelligence: a compendium, volume 115 of studies in computational intelligence* (pp. 851–880). Springer.

Joachims, T. (2005). A support vector method for multivariate performance measures. In L. De Raedt & S. Wrobel (Eds.), *Proceedings of the Twenty-second International Conference on Machine Learning (ICML 2009), volume 119 of ACM International Conference Proceeding Series* (pp. 377–384). ACM.

Jong, K., Marchiori, E., & Sebag, M. (2004). Ensemble learning with evolutionary computation: application to feature ranking. In X. Yao et al. (Eds.), *Parallel problem solving from nature – PPSN VIII, volume 3242 of lecture notes in computer science* (pp. 1133–1142). Springer.

Miikkulainen, R., Stanley, K. O., & Bryant, B. D. (2003). Evolving adaptive neural networks with and without adaptive synapses. *Evolutionary Computation, 4*, 2557–2564.

Krawiec, K., & Bhanu, B. (2007). Visual learning by evolutionary and coevolutionary feature synthesis. *IEEE Transactions on Evolutionary Computation, 11*(5), 635–650.

Lippmann, R., Haines, J. W., Fried, D. J., Korba, J., & Das, K. (2000). Analysis and results of the 1999 DARPA on-line intrusion detection evaluation. In H. Debar, L. Mé, & S. F. Wu (Eds.), *Recent advances in intrusion detection, volume 1907 of lecture notes in computer science* (pp. 162–182). Springer.

Liu, Y., Yao, X., & Higuchi, T. (2000). Evolutionary ensembles with negative correlation learning. *IEEE Transactions on Evolutionary Computation, 4*(4), 380–387.

Llorà, X., Sastry, K., Goldberg, D. E., Gupta, A., & Lakshmi, L. (2005). Combating user fatigue in igas: partial ordering, support vector machines, and synthetic fitness. In H.-G. Beyer & U.-M. O'Reilly (Eds.), *Genetic and Evolutionary Computation Conference (GECCO 05)*, ACM, pp. 1363–1370.

Margineantu, D., & Dietterich, T. G. (1997). Pruning adaptive boosting. *Proceedings of the Fourteenth International Conference on Machine Learning (ICML 1996)*, Morgan Kaufmann, pp. 211–218.

Mierswa, I. Evolutionary learning with kernels: a generic solution for large margin problems. In M. Cattolico (Ed.), *Genetic and Evolutionary Computation Conference (GECCO 06)*, ACM, pp. 1553–1560.

Mierswa, I. (2007). Controlling overfitting with multi-objective support vector machines. In H. Lipson (Ed.), *Genetic and Evolutionary Computation Conference (GECCO 07)*, pp. 1830–1837.

Mozer, M. C., Dodier, R., Colagrosso, M. C., Guerra-Salcedo, C., & Wolniewicz, R. (2001). Prodding the ROC curve: constrained optimization of classifier performance. *Advances in Neural Information Processing Systems*, NIPS, MIT Press.

Platt, J. (1999). Fast training of support vector machines using sequential minimal optimization. In B. Schölkopf et al. (Eds.), *Advances in kernel methods – support vector learning*. Morgan Kaufmann.

Poli, R. (2008). Genetic programming theory. In C. Ryan & M. Keijzer (Eds.), *Genetic and evolutionary computation conference, GECCO 2008*, (Companion), ACM, pp. 2559–2588.

Rosset, S. (2004). Model selection via the auc. *Proceedings of the Twenty-First International Conference on Machine Learning (ICML 2009), volume 69 of ACM International Conference Proceeding Series*. ACM.

Rumelhart, D. E., & McClelland, J. L. (1990). *Parallel distributed processing*. Cambridge: MIT Press.

Schapire, R. E. (1990). The strength of weak learnability. *Machine Learning, 5*, 197.

Schoenauer, M., & Xanthakis, S. Constrained GA optimization. In S. Forrest (Ed.), *Proceedings of the 5th International Conference on Genetic Algorithms*, Morgan Kaufmann, pp. 573–580.

Schölkopf, B., Burges, C., & Smola, A. (1998). *Advances in Kernel methods: support vector machines*. Cambridge: MIT Press.

Song, D., Heywood, M. I., & Nur Zincir-heywood, A. (2003). A linear genetic programming approach to intrusion detection. *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO)*, Springer, pp. 2325–2336, LNCS 2724.

Song, D., Heywood, M. I., & Nur Zincir-Heywood, A. (2005). Training genetic programming on half a million patterns: an example from anomaly detection. *IEEE Transactions on Evolutionary Computation, 9*(3), 225–239.

Sonnenburg, S., Franc, V., Yom-Tov, E., & Sebag, M. (Eds.). (2008). *Large scale machine learning challenge*. ICML Workshop.

Sutton, R. S., & Barto, A. G. (1998). *Reinforcement learning*. Cambridge: MIT Press.

Suttorp, T., & Igel, C. (2006). Multi-objective optimization of support vector machines. In Y. Jin (Ed.), *Multi-objective Machine Learning, volume 16 of Studies in Computational Intelligence* (pp. 199–220). Springer.

Tibshirani, R. (1996). Regression shrinkage and selection via the lasso. *Royal Statistical Society, B, 58*(1), 267–288.

Vapnik, V. N. (1995). *The nature of statistical learning*. New York: Springer.

Venturini, G., Slimane, M., Morin, F., & Asselin de Beauville, J. P. (1997). On using interactive genetic algorithms for knowledge discovery in databases. In Th. Bäck, (Ed.), *International Conference on Genetic Algorithms (ICGA)*, Morgan Kaufmann, pp. 696–703.

Zhang, T. (2003). Leave-one-out bounds for kernel methods. *Neural Computation, 15*(6), 1397–1437.

Nonstationary Kernels

►Local Distance Metric Adaptation

Nonstationary Kernels Supersmoothing

►Locally Weighted Regression for Control

Normal Distribution

►Gaussian Distribution

NP-Completeness

Definition

A *decision problem* consists in identifying symbol strings, presented as inputs, that have some specified property. The output consists in a yes/no or 0/1 answer. A decision problem belongs to the class P if there exists an algorithm, that is, a deterministic procedure, for deciding any instance of the problem in a length of time bounded by a polynomial function of the length of the input.

A decision problem is in the class NP if it is possible for every yes-instance of the problem to verify in polynomial time, after having been supplied with a polynomial-length *witness*, that the instance is indeed of the desired property.

An example is the problem to answer the question for two given numbers n and m whether n has a divisor d strictly between m and n. This problem is in NP: if the answer is positive, then such a divisor d will be a witness, since it can be easily checked that d lies between the required bounds, and that n is indeed divisible by d. However, it is not known whether this decision problem is in P or not, as it may not be easy to find a suitable divisor d, even if one exists.

The class of *NP-complete* decision problems contains such problems in NP for which if some algorithm decides it, then every problem in NP can be decided in polynomial time. A theorem of Stephen Cook and Leonid Levin states that such decision problems exist. Several decision problems of this class are problems on ►graphs.

Recommended Reading

Stephen Cook (1971). The complexity of theorem proving procedures. Proceedings of the third annual ACM symposium on theory of computing, 151–158.

Leonid Levin (1973). Universal'nye pereborne zadachi. *Problemy Peredachi Informatsii* 9(3): 265–266.

English translation, Universal Search Problems, in B. A. Trakhtenbrot (1984). A Survey of Russian Approaches to Perebor (Brute-Force Searches) Algorithms. *Annals of the History of Computing* 6(4): 384–400.

Numeric Attribute

Synonyms

Quantitative attribute

Definition

Numeric attributes are numerical in nature. Their values can be ranked in order and can be subjected to meaningful arithmetic operations. See ►Attribute and ►Measurement Scales.